NAIAD: A Dual-Purpose Bit-Slice Processor

R G SALIER, J MORRIS AND AHJ SALE
Department of Computer Science,
University of Tasmania,
GPO Box 252C, Hobart,
Tas 7001.

ABSTRACT

Using bit-slice components, a variable architecture processor has been designed to serve two purposes: to teach microprogramming and computer architecture to undergraduate students and to form the processing element of a dataflow machine. Microcode is loaded using a serial-protocol channel which can also be used for debugging: the current word in the MAP RAM and the pipeline register can be read at every clock cycle. The processor is provided with sufficient local memory (128K 32-bit words) to store test programs and data for teaching or a large segment of a dataflow program graph. In a dataflow processor, the design permits experimentation with operation granularity and provides a number of mechanisms for transferring program instructions to the processor - via the instruction stream, by loading new microcode and by transferring program segments to the local memory.

1. Introduction

The processor described in this paper was designed to meet two requirements: we wanted a microcoded processor for use in our Processor Architecture course and secondly as a processing element was needed for research into dataflow architectures.

1.1 Teaching Processor Architecture

Our processor architecture course contains (among other things) a segment in which the CISC and RISC design philosophies are studied by examining the architectures of a number of commercial CISC and RISC processors. It was decided that the essential differences between the two approaches would be better understood if students had some practical knowledge of microcoded machines. This led to a decision to build a processor with alterable microcode which could be used to demonstrate that the "architecture" of a machine is essentially contained in its microcode.

A number of others have embarked on similar projects, mainly using AMD 2901 style 4-bit slices[1,2,3,4,5], but now that 16-bit slices are available, we are able to produce a machine with a reasonable performance, avoiding the danger that students would regard the processor as "slow" or "old technology" and thus fail to be interested in the practical exercise.

communications network and the matching store. It is the processing element which we are constructing here. In its simplest form, the processing element simply takes an instruction, one or two data items and a destination node number, performs the requested operation to produce a result token which is despatched to the destination node.

Since the communications network and matching store represent significant bottlenecks - particularly in fine-grained dataflow machines, an aim of this project was to provide a processor whose granularity could be varied. Reducing the granularity implies giving away some potential parallelism[7], so a system which could assess this trade-off will contribute significantly to dataflow architecture research.

Another important requirement for a successful dataflow machine is to provide sufficient elasticity in the data paths to accommodate varying speeds in the various components of the machine.

Thus our requirements reduce to:

(a) The processor must be able to accept packets containing instruction, data and destination,

(b) it must be able to process the packets and pass the processed data packets to the communications network,

(c) it must be able to access the original program graph, so that instructions can be of the form "execute node x" where node x may involve a significant amount of computation as well as the fine-grain "this is a multiply and the operands follow" and

(d) the whole system must operate asynchronously with sufficient buffers between stages to minimize the idle time for all system elements.

3. Design

The design finally adopted uses two 16-bit ALUs forming a 32-bit computation unit, a 16-bit sequencer to control a 4K by 80-bit microcode RAM, a local memory for program, data and graph storage, an asynchronous input bus and FIFOs on the output bus. A block diagram of the system is shown in Figure 1 and individual sections are described in the following paragraphs.

3.1 Processor

The computation unit consists of 2 IDT49C402A 16-bit slices[1] connected together in carry-propagate mode. This registered ALU (RALU) provides 16 instructions and is controlled directly from the microcode. It provides 64 internal registers - enough to emulate the internal structures of most commercially available processors and obviate the need for additional external register memory.

---

[1]Originally we intended to purchase IDT 49C403 16-bit slices, because the serial protocol channel would have given us the ability to monitor the state of all the essential parts of the system in debug mode. These chips have been unavailable for some time, see the comments on industry directions later in this paper.

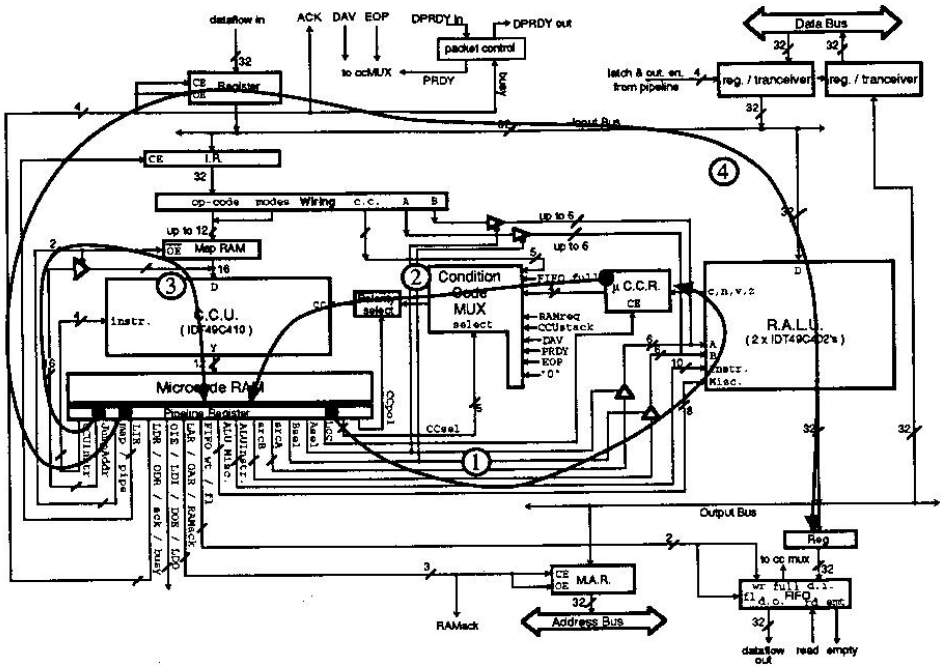

Figure 1 Naiad - Block Diagram

One problem in handling an arbitrary machine's instruction format is that the bits which specify registers containing operands are not in any "ISO standard" location in the instruction word. These bits can be somewhat painfully decoded from the instruction by microcode routines which treat the instruction as a piece of data and perform all the masks and shifts to extract the register indices. This, while satisfying our requirement that Naiad can emulate an arbitrary other processor is hardly likely to amuse our students. Also, it is not very realistic, in that any architect will have arranged for the instruction bits which select registers to be wired directly to the register selection logic. Since instruction sets often place the bits which select the registers that contain operands in fixed places in the instruction, we've provided two ALU register addressing alternatives:

(a) registers are addressed from microcode bits or

(b) instruction word bits can be fed via a "wiring loom" directly to the ALU register select inputs.

Thus the microcode (using bits Asel and Bsel) determines whether the ALU source and destination registers are obtained from the microcode directly (as they would be when selecting a fixed purpose register such as the PC or stack pointer) or from the instruction word sitting in the instruction register (IR).

3.2 Sequencer

The computer control unit (CCU) or microcode sequencer is placed in a conventional arrangement with MAP RAM, CCU, microcode RAM and pipeline register. The sequencer

used is an IDT 49C410A which provides a 16-bit microcode address as well as 16 internal operations - conditional and unconditional branches, loops and subroutine jumps. The next CCU instruction can be taken from the MAP RAM, internal counter or microcode.

For conditional branches, one of a total of 16 conditions is selected by microcode bits CCsel and passed to the CCU (after a possible polarity inversion - bit CCpol). The conditions which may be selected are:

Name	Source
Carry	ALU
Negative	"
Zero	"
Overflow	"
CCUStack	CCU Stack full
RAMReq	Local RAM Release Request
Extra (5)	Instruction Register
FIFO full	Output FIFO
DAV	Dataflow in handshake
PRDY	"
EOP	"
0	Hardwired

3.3 Microcode RAM

The microcode is a maximum of 80 bits: currently 78 are used. The allocation of microcode bits is shown in table 2. Initially microcode RAMs (IDT 71502 4Kx16 devices) providing a "serial protocol channel" (SPC) were selected to provide a simple means of loading the microcode and also reading the microcode word currently in the pipeline register as as an aid to debugging. These devices are also essentially obsolete *vide infra* so the system was re-designed using state-of-the-art 15ns access RAM devices which provided better performance. However a more complex microcode loading scheme, using registered SPC devices, was required.

3.4 External Interface

Each processor will be provided with the capability to act as a VMEbus slave[8]. The system controller (a 68010-based processor) on the VMEbus will provide a serial link to which a controlling terminal is attached and a debugger which can be used to "poke" small programs and data blocks into Naiad's memory for verifying microcode. It will also be used for examining results. As a dataflow processor, the controller's memory will serve as the main storage for program graphs.

Table 1. Microcode Structure

Name	Bits	Description
CCUInstr	4	Next CCU Instruction
JumpAddr	16	Address of next microcode word
Map/Pipe	2	Select source of next CCU instruction
LIR	1	Latch instruction register
DPktCtl	6	Data packet control
BusCtl	4	Memory data bus control
LAR/OAR	2	Latch, enable output of MAR
RAMAck	1	Release local memory for VMEbus access
BusAs	1	NAIAD/VMEBus memory switch
FIFOctl	2	Output FIFO control
ALUMisc	8	ALU shift and carry control
ALUInstr	10	ALU Instruction
Asrc	6	ALU A operand source
Bsrc	6	ALU B operand source
Asel	1	Select source of A operand address
Bsel	1	Select source of B operand address
Lcc	1	Latch condition code
CCpol	1	Condition code polarity
CCsel	5	Select condition code

3.5 Data flow

The central building block of the dataflow machine using Naiad is shown in Figure 2. Three processing elements are shown connected to a single matching store: this cluster has been designed so that additional processing elements can easily be added. As more processing elements are added, the capacity of the bus connecting the matching store and the PEs will be exhausted. The number of processing elements will need to be adjusted to balance the speed with which the matching store can handle tokens.

Figure 3 shows the mechanism for transferring data packets from a matching store to the processors in its cluster. When a set of data tokens is matched up , the matching store asserts the Data Packet Ready (DPRDY) signal which is propagated to the first idle processor. Elasticity in the link between the matching store and the processors will be provided by a FIFO in the matching store output stage. The first free processor will read a data packet from the matching store output FIFO using the DAV and ACK lines for handshaking. This link, like all links between all sections in our design, will be asynchronous to avoid clock distribution problems over a large machine[9].

Each processor has an output FIFO so that data tokens can be queued until the communications network is able to accept them. The communications network provides the link between processors and matching stores and also with the host. Input and output will be performed by transferring data tokens between the host and the communications network. The host will also be responsible for loading programs into the graph storage.

Partial graphs may then be transferred to individual processing elements as required. The local memory on each processor will then act as a read-only cache for program graphs.

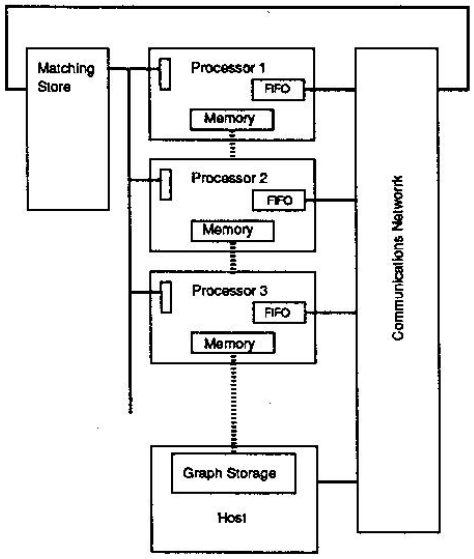

Figure 2. NAIAD matching store cluster. Three processing elements are shown - more could be installed in each cluster *see text*. The communications network would connect many such clusters.

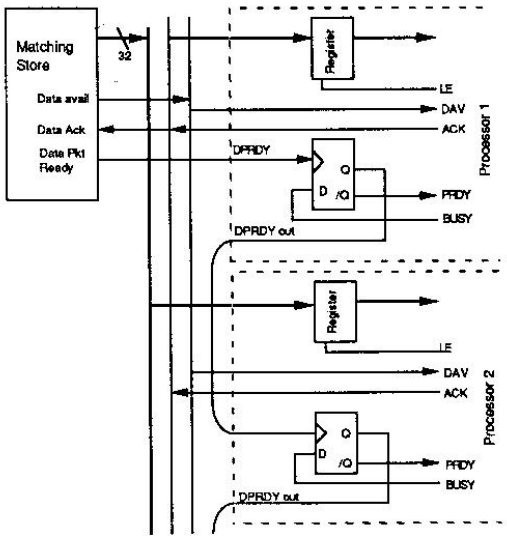

Figure 3 Matching store to processing element communication

Examination of the signal paths showed that four were critical in determining the maximum speed. Those four paths are shown in figure 1. In the assessments of each path which follow, the propagation delays and set-up times were obtained from manufacturer's figures.

Path 1 is the time from the beginning of a new clock cycle to valid output from the RALU. The pipeline register requires 12.5ns until it's output becomes valid. The register selection from the microcode is delayed by 6.5ns through the buffers before it reaches the RALU. Once the register select reaches the RALU, another 60ns is required until the Y output and the condition codes become valid. Finally, the condition code register requires a 2ns setup time before the condition codes can be latched on the rising edge of the next machine cycle. When all of these times are added, they indicate that path 1(a) requires a minimum of 80.5ns. The other paths may be similarly described.

Path 1(a) - ALU register select to valid condition codes

CP	--->	PR:Q	12.5	clock to pipeline register valid
PR:Q	--->	A & B	6.5	pipeline output to A&B select
A & B	--->	ALU:CC	60.0	ALU A&B to valid c.codes
c.code reg. setup			2.0	c.code register set-up time
			80.5 ns	

Path 1(b) - ALU instruction select to valid condition codes

CP	--->	PR:Q	12.5	clock to pipeline register valid
PR:Q	--->	ALU:cc	52.0	valid instr. bits to valid c.codes
c.code reg. setup			2.0	c.code register set-up time
			66.5 ns	

Path 2 - µCCR to new output address

CP	--->	CC:Q	7.2	clock to valid c.c.register output
CC:Q	--->	MUX:Y	11.0	c.c.register to valid MUX output
MUX:Y	--->	CC	8.5	MUX output to valid c.c. on CCU
CC	--->	CCU:Y	30.0	c.code valid to CCU valid Y
CCU:Y	--->	µC:D	15.0	RAM access
setup on pipeline reg.			2.5	
			74.2 ns	

Path 3 - Pipeline register to pipeline register

CP	--->	PR:Q	12.5	clock to pipeline register valid
PR:Q	--->	CCU:D	10.0	pipeline output to valid MAP output
CCU:D	--->	CCU:Y	20.0	input to CCU to valid CCU output
CCU:Y	--->	µC:DO	15.0	RAM access + µC reg. set-up time
setup on pipeline reg			2.5	
			60.0 ns	

Path 4 - Pipeline register to output FIFO data register

CP	--->	PR:Q	12.5	clock to pipeline register valid
PR:Q	--->	DI:Y	14.0	out.en. dataflow in to valid data
DI:Y	--->	ALU:Y	51.0	ALU A&B to valid c.codes
MAR/MDR/DFOR setup			4.0	c.code register set-up time
			82.5 ns	

Abbreviations used in the path descriptions

CP	clock pulse. Rising edge of next machine cycle
PR:Q	Pipeline Register output
A & B	register select input to RALU
ALU:CC	condition code outputs from ALU
CC:Q	condition code register output
MUX:Y	output of c.code multiplexer
CC	condition code input of CCU
CCU:Y	Y output address of CCU
µC:DO	Microcode Data Out. Used for time though the microcode RAM, including set-up time for the pipeline register
DI:Y	dataflow-input-register output

Path 4 requires the longest time (82.5ns), and hence this becomes the minimum cycle time for the machine - corresponding to a clock rate of 12.1MHz. Any reduction of propagation or set-up delays within this path directly affect the speed of the whole machine, hence the fastest chips available were used.

One possible improvement may be obtained by noting that not all ALU operations require a full 60ns. Thus we have the option of running the processor faster for some operations and inserting extra microcode words when necessary. This may produce faster average throughput - at the expense of microcode complexity. (In general, it is unlikely that the additional cycle will be able to do anything useful in parallel with the wait for the condition codes to be available.)

It should be noted that a large contribution to the critical paths comes from inter-chip delays. Were all devices freed of the need to drive pF loads coming from pins and PCB tracks - as in a single chip CISC processor - a much faster cycle time would become possible.

4. Directions in Architecture

We experienced a number of problems in obtaining components for this project which seem to indicate a significant direction "shift" in computer architecture. A few years ago, building a bit-slice processor was the strategy of choice when a processor for a specific purpose was needed: it enabled you to push your critical operations into specially microcoded instructions speeding up overall throughput by cutting down instruction fetches and data accesses. With the number of transistors per die now routinely exceeding 10^6, it is possible to build a complete processor with cache and a significant number of additional components (eg FPU, MMU) on a single chip. Since all the critical components are on the same piece of silicon, overall processor throughput has increased to the point

where it is not possible to match such speeds with designs that contain critical elements spread over a number of chips.

Despite using components which have potential clock speeds up to 40MHz, we were unable to run our whole system at greater than 12MHz. This was due to accumulated delays through individual chips in the critical paths. Thus only an extremely specialized design - one which could use the ALU alone - could even make use of the potential present there. Unfortunately, given the limited instruction sets of the currently available devices, they are extremely limited without microcode to drive them!

A state-of-the-art RISC processor, although it may not be optimized for a target application, is likely to be faster overall. As a consequence, the demand for bit-slice components would seem to be ebbing fast and it is likely that most will disappear from suppliers' catalogues in the near future. This may be seen as a blow to the research community as it will force much experimental architecture research into the custom VLSI arena. Even with silicon compilers we are still a few years away from the point where a student can design a significant processor of the complexity of Naiad in six months - as was done here.

5. Summary

The Naiad processor design effort has shown that it is possible to design a variable architecture system to meet both our initial requirements. The performance predicted for Naiad is near to the maximum we can expect to obtain with "off-the-shelf" bit-slice technology. We do not expect to see any further development in this area from commercial manufacturers. Some difficulties experienced attempting to obtain components for this project has highlighted a significant trend in the design of special purpose high-performance systems: if state-of-the-art performance is desired, it seems that full-custom VLSI may soon be the only option available. Very large re-programmable gate arrays may provide a partial solution, but we must all hope that the cost of custom VLSI continues to drop!

6. References

1. J A Black, C D Isler, J N Gowdy, *Design and Implementation of a Bit-slice Computer Based on the AM2903 CPU,* Proceedings of the 14th Southeastern Symposium on System Theory, Blacksburg, VA (1982) 132.
2. V O Blackledge, D C Bolles, *An Educational Bit-slice Computer,* First Annual Phoenix Conference on Computers and Communications, Phoenix, Arizona (1982) 327.
3. D K DuBose, D K Fotakis, D Tabak, D, *A Microcoded RISC,* Computer Architecture News **14** (1986) 5.
4. I M Longair, A D Morgan, A D, *Bit-slice Microprocessors - a Hardware and Software Development Kit,* International Journal of Electrical Engineering Education (UK) **24** (1988) 325.
5. H Namekawa, T Kishigami, *An Experimental Small Computer for Microprogramming Courses,* Ibaraki University Engineering Faculty Research Reports, **33** (1985) 205 (in Japanese).

6. A H Veen, *Dataflow Machine Architecture,* ACM Computing Surveys, **18** (1986) 365 contains a review of much of the early work.

7. V P Srini, *An Architectural Comparison of Dataflow Systems,* Computer, **19** (1986) 68.

8. VMEbus International Trade Association, *The VMEbus Specification,* (VITA, 1987). This publication contains ANSI/IEEE STD 1014-1987.

9. K Hiraki, S Sekiguchi, T Shimada, *System Architecture of a Dataflow Supercomputer*, TEN CON 87, Seoul (1987) 1044.